VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Hra Carcassonne - Rozšíření Věže a rozšíření Princezna a drak

Game Carcassonne - Extension Tower and extension Princess and Dragon

2016 Pavel Žůrek

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Pavel Žůrek**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Hra Carcassonne - Rozšíření Věže a rozšíření Princezna a drak
Game Carcassonne - Extension Towers and Extension Princes and Dragon

Jazyk vypracování: čeština

Zásady pro vypracování:

Cílem práce je rozšířit stávající verzi hry Carcassonne v jazyce Java o další dvě rozšíření (Věže, Princezna a drak). Rozšíření Věže dovolí stavět v krajině nové věže a zajímat soupeřovy figurky. Rozšíření Princezna a drak umožní využít nových figurek Draka a Víly a jejich pohyb dle pravidel. Hra také zahrne nové karty s princeznou, které mohou odstranit rytíře z měst. Výsledná aplikace bude dovolovat při vytváření nové hry libovolně zapínat a vypínat tyto dvě rozšíření.

Implementace hry Carcassonne bude:
1. Umožňovat hraní hry Carcassonne a jejich dvou rozšíření (Věže, Princezna a drak)
2. Umožňovat volby jednotlivých rozšíření, které budou použity ve hře.
3. Obsahovat přepracovaný způsob vykreslování herního pole.
4. Umožňovat zvýraznění souvislé oblasti mapy (cesta, město, louka, ...).
5. Pro přihlášení uživatelů bude využit nově vznikající herní portál.

Práce bude obsahovat:
1. Implementaci rozšíření Věže a rozšíření Princezna a drak hry Carcassonne.
2. Programátorskou dokumentaci řešení s využitím diagramů jazyka UML.
3. Uživatelskou dokumentaci aplikace.

Seznam doporučené odborné literatury:

[1] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides (Gang of Four): Návrh programů pomocí vzorů. Grada. Praha 2003. ISBN 8024703025
[2] DARWIN, Ian F. Java cookbook. 2nd ed. Sebastopol, CA: O'Reilly, c2004, xxiv, 829 p. ISBN 05-960-0701-9. Dostupné z: http://it-ebooks.info/book/2249/
[3] Carcassonne (hra). In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2012-10-05]. Dostupné z: http://cs.wikipedia.org/wiki/Carcassonne_(hra)

Dále podle pokynů vedoucího bakalářské práce.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. David Ježek, Ph.D.**

Datum zadání: 01.09.2014

Datum odevzdání: 29.04.2016

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 29. dubna 2016 .....

Souhlasím se zveřejněním této bakalářské práce dle požadavků čl. 26, odst. 9 Studijního a zkušebního řádu pro studium v bakalářských programech VŠB-TU Ostrava.

V Ostravě 29. dubna 2016 .....

Rád bych na tomto místě poděkoval všem, kteří mi s prací pomohli. Především svému vedoucímu Ing. Davidu Ježkovi za jeho trpělivost. Ale také rodičům a celé rodině za podporu po celou délku studia, i kolegům v práci za rady.

Abstrakt

Tématem mé bakalářské práce je rozšíření již existující hry Carcassonne o rozšíření Věže a Princezna a drak. Hra je implementována v jazyce Java a využívá technologie RMI pro komunikaci se serverem a Java WebStart pro spouštění. Nejdříve jsou nastíněna základní pravidla hry. Poté následují pravidla, která jsou přidána s novými rozšířeními. Vzhledem k tomu, že programovací jazyk a použité technologie už byly odůvodněny v předchozích bakalářských pracích, další část je věnována výběru vývojového prostředí a dalších programů vhodných pro programování větších projektů. Dále je popsáno samotné řešení problémů při implementaci a implementace samotné. Tato práce navazuje na sérii bakalářských prací, řešících hraní her online a základní hry Carcassonne s rozšířeními Hostince a katedrály a Kupci a stavitelé.

Klíčová slova: Bakalářská práce, on-line hra, Carcassonne, Remote Method Invocation (RMI), Java web start (JWS), programovací jazyk Java

Abstract

The topic of my bachelor thesis is an extension of an existing game Carcassone. Extensions are Tower and Princess and Dragon. The game is implementated in Java language and it uses RMI technology for communication with server and Java WebStart for launching. At the begining there are basic rules of the game. Then the rules for added extensions follow. Whereas the programming language and used technologies were justified in previous bachelor thesis, next part is about development enviroment and other programs suitable for programming larger projects. Then the implementation problems and implementation itself is described. This thesis is continuation of series of bachelor thesis, that desribe online game playing and basic Carcassonne game with extensoins Inns and cathedrals and Buyers and builders.

Key Words: Bachelor thesis, on-line game, Carcassonne, Remote Method Invocation (RMI), Java web start (JWS), programming language Java

Obsah

Seznam použitých zkratek a symbolů **10**

Seznam obrázků **11**

1 Úvod **13**

2 Pravidla hry **14**
2.1 Základní hra 14
2.2 Věže 15
2.3 Princezna a drak 17

3 Příprava na implementaci **20**
3.1 Vývojové prostředí 20
3.2 Verzovací systém 21
3.3 Issue tracking (Systém pro sledování problémů) 21

4 Výchozího stav aplikace **23**
4.1 String 23
4.2 Opakování kódu 23
4.3 Pole polí 25
4.4 Souřadnice karty 25
4.5 Změna velikosti 26
4.6 Zobrazení figurek 26
4.7 Práce s obrázky 27
4.8 Další možná vylepšení 28

5 Implementace **29**
5.1 Zvýraznění souvislé oblasti mapy 29
5.2 Rozšíření Věže 31
5.3 Rozšíření Princezna a drak 33

6 Analýza systémem SonarQube **37**

7 Závěr **38**

Literatura **39**

Přílohy **39**

A CD **40**

B Uživatelská dokumentace **41**
B.1 Založení hry 41
B.2 Připojení do hry 41
B.3 Herní rozhraní 42

Seznam použitých zkratek a symbolů

RMI	–	Remote Method Invocation
JWS	–	Java WebStart
IDE	–	Integrated Development Environment
GUI	–	Graphic User Interface

Seznam obrázků

1 Ukázka kartiček krajiny [1] . . . 14
2 Ukázka umístění figurek [1] . . . 15
3 Červený hráč umístí dílek věže a může uvěznit modrou figurku na kartě 2, nebo 3. 16
4 Modrý hráč umístí druhé patro věže. Nyní si může vybrat, zda uvězní zelenou figurku na kartě číslo 1, nebo žlutou figurku na kartě číslo 7. . . . 16
5 Příklad karty s vyobrazením sopky. . . . 17
6 Příklad: Červený přiloží kartu se symbolem draka a položí na ni svou figurku. Poté přesune draka o pole nahoru. Modrá figurka je vrácena majiteli. Modrý hráč a po něm opět červený posunou draka ve směru šipek. Potom už drak nemá kam jít. . 18
7 Příklad: Červený přiloží k městu kartu se symbolem princezny. Nyní může jednu figurku vrátit do zásoby majiteli. Logicky vrátí modrou a tím dostane všechny body za uzavřené město. . . . 19
8 Původní figurky (nahoře) a nové figurky (dole) . . . 27
9 Tvar zvýraznění oblastí . . . 29
10 Třídní diagram uložení karet hracího pole . . . 30
11 Ukázka zvýrazněné louky . . . 30
12 Ukázka zvýrazněné cesty . . . 31
13 Diagram algoritmu zvýraznění oblasti . . . 31
14 Věž ve hře . . . 32
15 Formulář pro vykoupení figurky . . . 33
16 Sekvenční diagram pohybu draka . . . 36
17 Část výstupu z analýzy systémem SonarQube (verze 1.0 je původní, verze 2.0 je nynější) . . . 37
18 Založení hry . . . 41
19 Připojení do hry . . . 42
20 Herní rozhraní . . . 43
21 Umístění figurky . . . 44

Seznam výpisů zdrojového kódu

1 Část původní metody AddFigure.execute 24
2 Ukázka rozdílu po použití objektu StavHry 25
3 Metoda getX enumu WorldSide 26
4 Přemosťující metoda třídy PossibleFigures 34
5 Abstraktní metody třídy ExtendedMove 35
6 Implementované metody třídy ExtendedMove 35

1 Úvod

Carcassonne je moderní společenská hra německého typu. Jejím autorem je Klaus-Jürgen Wrede.

Jde o stolní hru pro 2 až 6 hráčů založenou na postupném přikládání jednotlivých kartiček. Tím vzniká krajina s cestami, městy, kláštery a loukami, kterou si hráči obsazují svými figurkami. Hra získala v roce 2001 první místo v anketě Spiel des Jahres (Hra roku) a Deutscher Spiele Preis (Hra německé veřejnosti). [4]

Nová rozšíření do hry přidávají figurky draka, víly a věží, čímž hru okoření převážně o zvraty ve zvolené strategii.

Rozhodl jsem se pro tuto práci, protože jsem fanouškem této hry už od dětství a sám vlastním 5 rozšíření a 4 mini-rozšíření.

2 Pravidla hry

Jihofrancouzské město Carcassonne je pověstné svým jedinečným opevněním, které se vyvíjelo od římských dob až po časy rytířů. Hráči se vydávají se svou družinou na cesty a louky, do měst a klášterů, aby hledali štěstí kolem Carcassonne. Vzhled krajiny je v jejich rukou a strategické rozmístění družiníků, ať už sedláků, lupičů, mnichů nebo rytířů, určuje cestu k úspěchu.

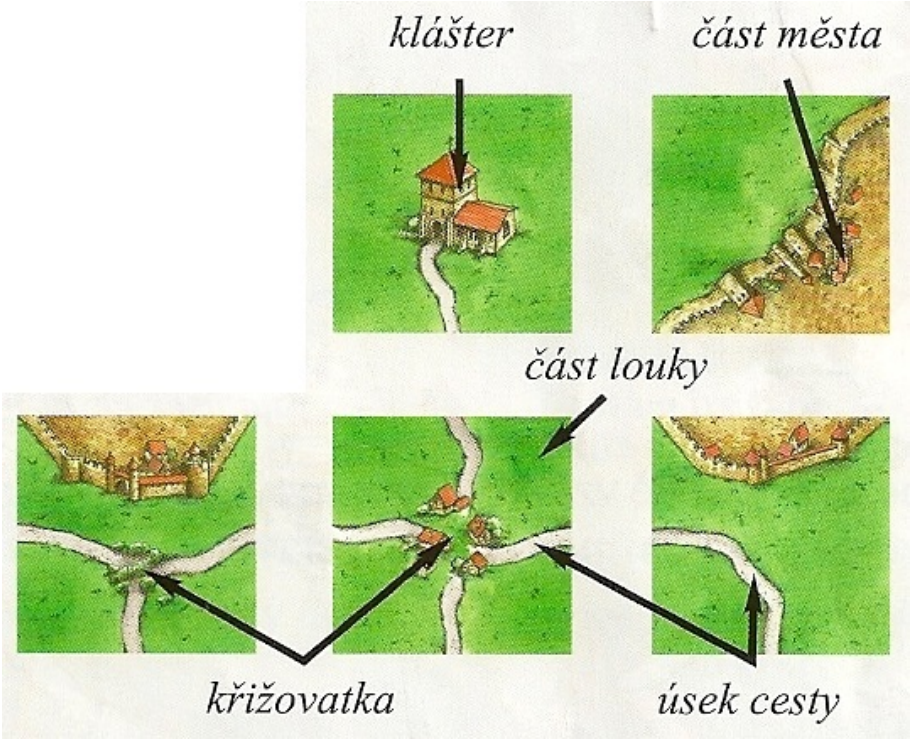
Hráči vykládají jeden po druhém karty s motivem krajiny. Vznikají cesty, města, louky a kláštery, na které mohou hráči umístit své figurky, aby sbírali body. Body se započítávají jak v průběhu hry, tak na jejím konci, a proto je vítěz znám až po konečném sčítání.

Tak jako je jihofrancouzské město Carcassonne pověstné svým jedinečným opevněním, tak se hra Carcassonne stává pověstnou svou úspěšností. Od jejího uvedení na trh v roce 2001 se prodalo již několik milionů kusů.

K dispozici je řada rozšíření, která hru obohacují o nové prvky. [1]

2.1 Základní hra

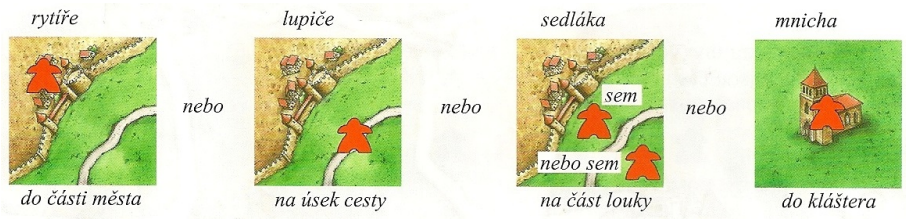
Informace o pravidlech jsem čerpal z oficiálních pravidel [1]. Hra je určena pro 2-5 hráčů. Obsahuje celkem 40 figurek pro vykládání na herní plochu a 72 karet s motivy krajiny. Na kartách jsou části měst a luk, úseky cest, křižovatky a kláštery.



Obrázek 1: Ukázka kartiček krajiny [1]

Hráči vykládají jeden po druhém karty s motivem krajiny tak, aby vznikala souvislá krajina. Vznikají cesty, města, louky a kláštery, na které mohou hráči umístit své figurky, aby sbírali body. Body se započítávají jak v průběhu hry, tak na jejím konci, a proto je vítěz znám až po závěrečném vyhodnocení.

Po vložení karty může hráč (pokud není dáno jinak rozšířením nebo není území již obsazené) položit na právě umístěnou kartičku jednu svou figurku.



Obrázek 2: Ukázka umístění figurek [1]

Pokud je území uzavřeno (klášter je uzavřen pokud je kartička obestavěna osmi dalšími kartami), hráč si figurku bere zpět do své zásoby a připočte si odpovídající počet bodů. Pokud má na území své figurky více hráčů a jejich počet je stejný, dostanou všichni hráči plný počet, pokud má některý hráč na území figurek více, získává plný počet pouze on.

Jediné území, které nelze uzavřít, je louka. Figurky na louce zůstávají do konce hry.

Na konci hry se pak započítají louky a neuzavřená území.

2.2 Věže

Rozšíření Věže, jak název napovídá, do hry přidává figurky věží, díky kterým mohou hráči zajmout figurku protihráče. Tu pak majitel může získat zpět. Přijde ovšem o pozici, na které figurka byla.

2.2.1 Umístění věže

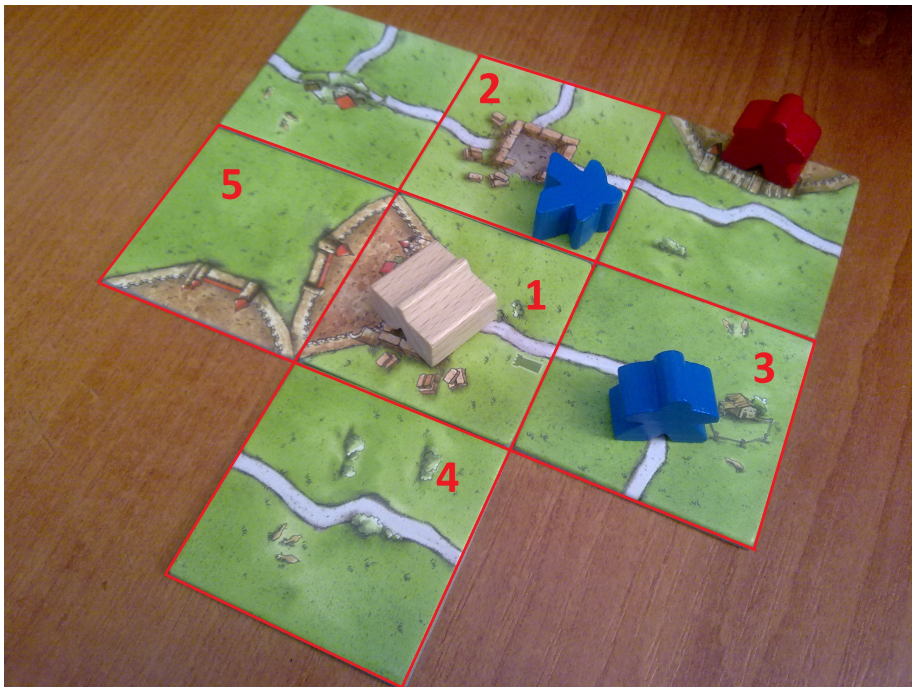
Po umístění karty do hry, podle základních pravidel, může hráč buď umístit svou figurku podle klasických pravidel, nebo využít nové možnosti:

- Umístit jeden ze svých dílků věže na kteroukoli kartu krajiny se základy věže
- Umístit jeden ze svých dílků věže na kteroukoli již rozestavěnou věž
- Umístit jednu svou figurku na kteroukoli věž a tím zabránit další stavbě této věže

2.2.2 Umístění dílku věže a uvěznění figurky

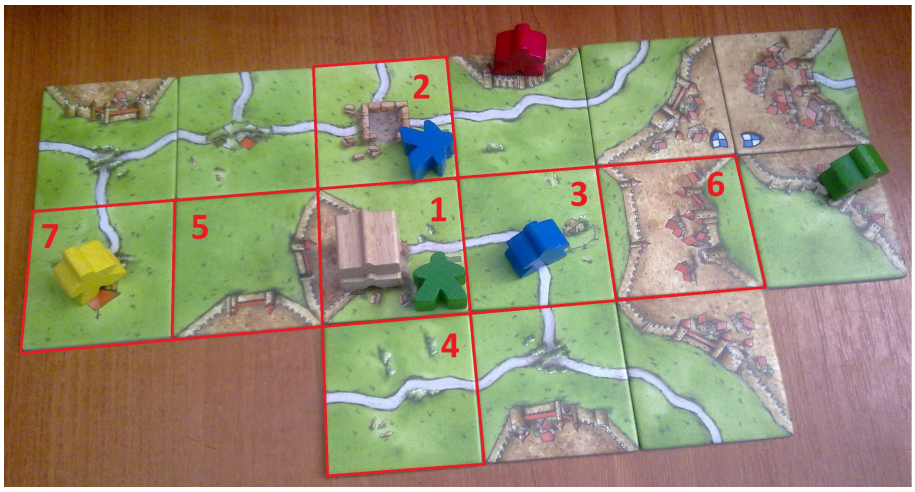
Kdykoliv hráč umístí dílek věže, okamžitě může uvěznit jednu figurku kteréhokoliv z ostatních hráčů (dříve než se na konci kola započítají body). Figurky, které je možno uvěznit, určíme podle výšky právě stavěné věže. Uvězněnou figurku si vezme hráč k sobě.

Když je postaveno první patro věže, hráč může uvěznit figurku z karty, na které je věž, nebo z jedné ze čtyř sousedních karet.



Obrázek 3: Červený hráč umístí dílek věže a může uvěznit modrou figurku na kartě 2, nebo 3.

Hráč, který postaví druhé patro věže, může uvěznit figurku z jednoho z devíti možných polí. S každým patrem navíc se mohou hráči naskytnout čtyři další možnosti ve čtyřech směrech. Tyto možnosti nemohou přesahovat mezery v krajině. Výška věží není nijak omezena.



Obrázek 4: Modrý hráč umístí druhé patro věže. Nyní si může vybrat, zda uvězní zelenou figurku na kartě číslo 1, nebo žlutou figurku na kartě číslo 7.

2.2.3 Umístění figurky na vrchol věže

Hráč může umístit jednu ze svých figurek na vrchol některé z věží. Od této chvíle je tato věž zablokovaná a není možné ji dále stavět. Figurka zůstává na věži až do konce hry, pokud není zajata z jiné věže, nebo vrácena do zásoby majiteli drakem. Takto může hráč ochránit svou figurku před zajetím.

2.2.4 Uvězněné figurky

Hráč může svou zajatou figurku získat zpět dvěma způsoby. V případě, že dva hráči mají navzájem v zajetí figurku toho druhého, automaticky si je musí hned vyměnit. Ve svém tahu může hráč navíc od jiného hráče vykoupit figurku ze zajetí za 3 body (vykupujícímu hráči se odečtou a věznitelí se přičtou). Tato figurka může být už v tomto kole použita.

2.3 Princezna a drak

V balení tohoto rozšíření můžeme najít dvě nové postavičky, jsou to drak a víla. Drak se pohybuje po herním plánu a pojídá figurky. Víla může drakovi zabránit ve vstupu na jedno políčko a přidává body. Do hry přibývají i symboly na kartách. Symbol princezny umožňuje "vyhostit"z hradu figurku a tajná chodba dovolí umístění figurky na neobsazenou plochu.

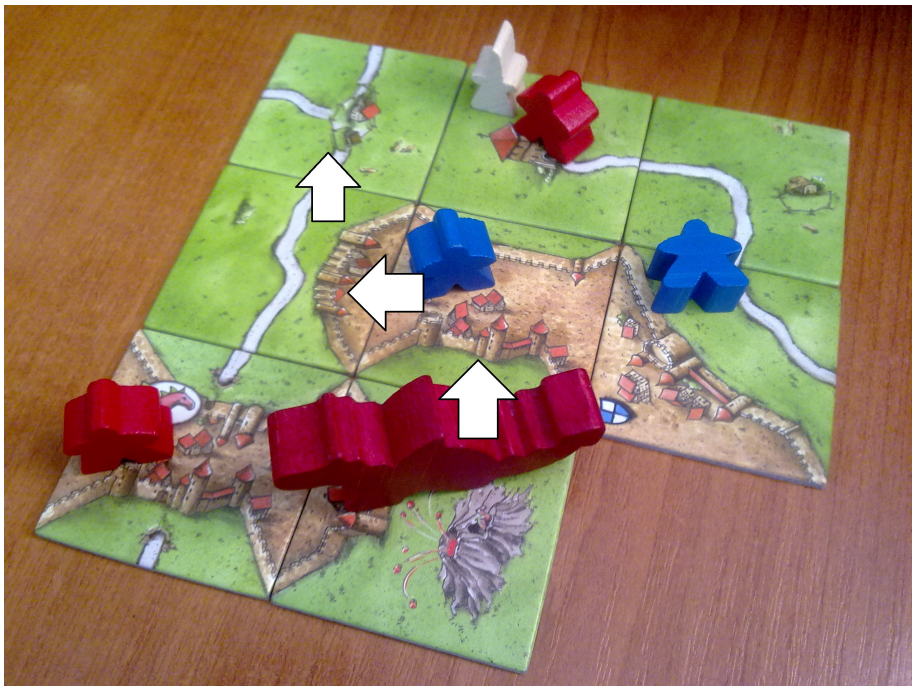
2.3.1 Drak

V zásobě karet je několik karet krajiny s obrázkem sopky. Po umístění takové karty hráč ve svém kole již nemůže umístit žádnou figurku a musí na novou kartu se sopkou přesunout figurku draka.

Obrázek 5: Příklad karty s vyobrazením sopky.

Po vytažení kartičky se symbolem draka se drak začne pohybovat po herním plánu. Pokud dosud nebyla vytažena sopka, karta se odloží a po umístění sopky se zamíchá zpět do zásoby. Drak se pohne maximálně o šest polí. Každý pohyb o jednu kartu provádí hráči postupně, podle pořadí hry. První, kdo hýbe drakem, je hráč, který si vylosoval kartičku s dračím symbolem. Drak může vstoupit v jednom svém kole na každé pole maximálně jednou a nesmí vstoupit na kartičku, na které je umístěna figurka víly. Pokud se drak nemá kam posunout, jeho tah končí.

Pokud drak při pohybu vstoupí na kartičku, na které jsou umístěny figurky, drak je "sežere"(včetně figurek na věži) a jsou vráceny majiteli.



Obrázek 6: Příklad: Červený přiloží kartu se symbolem draka a položí na ni svou figurku. Poté přesune draka o pole nahoru. Modrá figurka je vrácena majiteli. Modrý hráč a po něm opět červený posunou draka ve směru šipek. Potom už drak nemá kam jít.

2.3.2 Víla

Pokud hráč může ale neumístí ve svém kole figurku na hrací plochu, může umístit nebo přemístit postavičku víly, a to na kteroukoliv kartu na hrací ploše, kde již nějakou svou figurku má. Na kartu s vílou nesmí vstoupit drak. Pokud víla na začátku tahu hráče stojí na jedné kartě s jeho figurkou, dostává hráč ihned jeden bod. Při uzavření oblasti, v níž se nachází figurka na jedné kartě s vílou, jsou majiteli oblasti připsány další tři body.

2.3.3 Princezna

Na některých kartách s hradem je symbol princezny ve věži. Po přiložení této karty do města, ve kterém se nachází jeden nebo více rytířů, musí hráč na tahu určit, který rytíř (včetně jeho vlastních) bude vrácen do zásoby majiteli. Hráč na tahu v tomto případě nesmí umístit žádnou svou figurku do hry.

Pokud hráč přiloží kartu se symbolem princezny do neobsazeného města, může svou figurku umístit.

Symbol princezny se vztahuje pouze na jedno konkrétní město, na kterém je symbol umístěn.

Obrázek 7: Příklad: Červený přiloží k městu kartu se symbolem princezny. Nyní může jednu figurku vrátit do zásoby majiteli. Logicky vrátí modrou a tím dostane všechny body za uzavřené město.

2.3.4 Tajná chodba

Když hráč umístí do hry kartu se symbolem tajné chodby, může umístit svou figurku do kterékoliv neobsazené, neuzavřené oblasti (kromě louky).

3 Příprava na implementaci

Pro vývoj jakéhokoliv většího než malého projektu je velmi vhodné použít vyhovující vývojové prostředí a další programy pro ulehčení a zpřehlednění práce.

3.1 Vývojové prostředí

Vývojové prostředí je nejdůležitější nástroj programátora. Nemusí to být nutně přímo program určený pro psaní zdrojových kódu, nýbrž se může jednat i o obyčejný textový editor. Moderní IDE ale nabízejí obrovskou škálu podpůrných funkcí od barevného zvýrazňování klíčových slov, až po nápovědu, kontrolu syntaxe a automatickou kompilaci. Většina dnešních programátorů si život bez těchto nástrojů neumí představit.

Při svých pracích jsem dříve pro programování v jazyce Java používal převážně 2 IDE, Eclipse a NetBeans. Z mého pohledu byl výběr docela jasný, rád bych ho ale zdůvodnil.

3.1.1 Eclipse

Vývojové prostředí Eclipse jsem začal používat ve škole pro vytváření svých prvních projektů v Javě, protože většina ostatních studentů i učitelů ho používala. Byl jsem z tohoto programu zmatený a nemohl jsem se v něm vyznat, proto jsem začal hledat náhradu.

Výhody:

- vyšší rychlost
- modifikovatelnost

3.1.2 NetBeans

Toto IDE jsem začal používat již na střední škole pro programování jednoduchých webových aplikací v PHP. Po svém neúspěchu s Eclipsem jsem se rozhodl ho vyzkoušet i později na VŠ. Od té doby používám právě NetBeans nejen ve škole, ale i v práci.

Výhody:

- větší přehlednost
- vestavěný GUI designer

3.1.3 Výběr

S přihlédnutím k mým zkušenostem a znalostem jsem se rozhodl pro IDE NetBeans.

3.2 Verzovací systém

Pro zpřehlednění postupu práce je velmi vhodné použít verzovací systém. Je to nástroj, který umožňuje se vracet nebo porovnávat nové části kódu s těmi starými. Nabízí i další funkce, převážně pro práci v týmu. Ovšem i pro osamoceného programátora je výhodné ho použít.

3.2.1 Subversion (SVN)

Subversion je systém pro správu verzí. Data jsou na serveru (repozitář), ze kterého si stáhneme soubory na svůj disk (pracovní kopie). Po vykonání změn, zpravidla to bývá po dokončení nějaké funkční části, se provádí "commit". To je příkaz, kterým data odešleme na server, kde je vidí ostatní v týmu. Ke každému commitu se přidává zpráva o tom, co programátor změnil.

Tento systém je nejstarší verzovací systém, který znám. Bohužel, jde to na něm trochu poznat a ze zkušenosti vidím, že lidé ho opouštějí ve prospěch nového a moderního GITu.

3.2.2 GIT

Git už není centralizovaný, ale distribuovaný verzovací systém. Můžeme například provádět commit bez připojení k internetu, resp. k serveru s repozitářem. Tím je nám umožněno provést i více commitů, než je odešleme na server (tzv. push). Daleko jednodušší je také práce s větvemi (branch). Každý repozitář může mít více větví, které se navzájem neovlivňují, a tak můžeme například vytvářet dvě verze softwaru, kde jedna může mít více funkčnosti.

S verzovacím systémem GIT jsem se seznámil na stáži, kde se mi velice zalíbila jeho jednoduchost a zároveň komplexnost. Nabízí například jednoduché slučování různých úprav projektu, nebo takzvaný "blame"režim, díky kterému můžeme přesně vidět, který člen týmu napsal který řádek kódu.

3.2.3 Výběr

I přes to, že rozdíl mezi SVN a GITem pro mě není tak razantní, s druhým zmiňovaným systémem mám více zkušeností a možnost commitu bez přístupu k síti, což je užitečná vlastnost. Proto jsem si zvolil systém GIT.

3.3 Issue tracking (Systém pro sledování problémů)

Takovýto systém se může zdát zbytečný, ale z mého pohledu je to výborný nástroj. Je důležité, aby programátor nezapomněl vyřešit chybu, kterou objevil ve svém kódu. Ne vždy ji ale může odstranit ihned, často ji ani odstranit nepotřebuje, dokud nedokončí rozdělanou práci. Případně pokud pracuje v týmů, nemusí být žádoucí, aby opravoval chyby ostatních. Samozřejmě neslouží jen pro sledování chyb, ale pro zápis nových úkolů a jejich organizaci podle priority a typu.

3.3.1 Bitbucket

Bitbucket je webová služba nabízející verzovací nástroj GIT nebo Mercurial, a k němu i issue tracker.

Rozhodl jsem se ho použít, protože ho znám již dlouhou dobu a jsem s ním spokojený.

3.3.2 GitHub

Další služba konkurující Bitbucketu je například GitHub, který nabízí verzování výhradně GIT.

4 Výchozího stav aplikace

Před implementací jsem samozřejmě musel zjistit, jak aplikace funguje. Pochopení cizího kódu většinou nebývá jednoduché, ani velká část tohoto nebyla výjimkou. Některé části kódu bylo potřeba upravit, ať už kvůli méně vážným chybám, jako je například zbytečné opakování kódu, nebo kvůli závažnějším, jako je použití krajně nevhodných datových typů. Některé z těchto chyb bych rád zmínil, protože jejich řešení bylo nakonec velmi podstatnou částí mé práce.

4.1 String

String je datový typ, který se používá k uložení textů. Z programátorova hlediska není dobré ho požívat tam, kde opravdu není potřeba. Například pro uložení světové strany, typu figurky, typu oblasti, nebo symbolu na kartičce. Nevýhodou Stringu je například špatná kontrola při použití nedefinované, nebo neočekávané hodnoty, což může být byť jen překlep.

Tento problém se dá vyřešit změnou na výčtový typ enum. Ten může nabývat jen předem definované hodnoty. Každá hodnota odpovídá jedné instanci třídy. Navíc může mít vlastnosti (u světových stran například souřadnice, u typu figurky pak cestu k souboru s jejím obrázkem). Další výhodou je rychlejší a přehlednější porovnávání, protože můžeme použít klasické `"=="`místo metody `"equals"`.

V celé aplikaci jsem proto eliminoval výskyt datového typu String a v množství případů jsem ho nahradil svým enumem. Vznikly proto tyto nové enumy:

- WorldSide - Světové strany (North, East, South, West, NorthEast, NNW atd.)
- FigureType - Typ figurky (Basic, Pig, Large, Builder, Tower, Fairy, Dragon)
- RegionType - Typ oblasti (City, Grass, Church, Road, Tower)
- ExtensionType - Typ rozšíření
- CardSymbol - Symboly vztahující se k celé kartě (Dragon, Volcano, Tunel)
- RegionSymbol - Symboly vztahující se k oblasti (Erb, Corn, Princess atd.)
- Side - Strany, do kterých zasahuje region Grass (None, Left, Right, Both)

4.2 Opakování kódu

Java je objektový jazyk, což znamená, že kód je rozdělen do menších, navzájem propojených částí. Každá taková část (objekt) se skládá z dat (atributy) a možných operací nad nimi (metody). Objekty mezi sebou mohou dědit metody a tím můžeme omezit množství kódu, který by se jinak zbytečně opakoval. Popřípadě je možné vytvořit statickou třídu, jejíž instance nemají vlastní data, ale jsou společná pro všechny instance třídy. Uvnitř obyčejné třídy je také možné vytvořit

statickou metodu, nebo proměnnou, takové metody opět nemají přístup k atributům objektu. Proto je jejich volání dobré provádět nad třídou a nikoliv nad objektem.

V původní aplikaci bylo několik tříd s jedinou statickou metodou, která umisťovala figurky různého typu na kartu. Každá z těchto metod měla skoro tři stovky řádků a kód se zhruba každých deset řádků opakoval jen s jednou změněnou proměnnou. Metoda přijímala souřadnice, na které hráč na náhledu karty kliknul, a data hry.

V ukázce kódu 1 vidíme jen malou část původní metody. Tento úsek kódu se 17 krát v podstatě opakoval. Co se měnilo byla první podmínka, kdy souřadnice odpovídaly jinému výčtu `WorldSide` (úprava by šla ale provést, i kdyby byly světové strany uloženy stále jako String). Ve zbytku se měnila už jen světová strana, odpovídající daným souřadnicím.

V konstruktoru třídy `Figure` také můžeme vidět původní String, označující figurku ze základní části hry. Nyní by konstruktor místo `"normal"` obsahoval `FigureType.BASIC`.

Na následujícím řádku lze taky vidět, jakým způsobem se provádělo odečítání figurek ze zásoby hráče. V současném stavu má hráč HashMapu, kde klíčem je typ figurky a hodnotou jejich počet v jeho zásobě. Přidal jsem také novou metodu `removeFigure`, která přijímá v argumentu typ figurky. Navíc vrací `boolean` hodnotu, jestli byla figurka odečtena. Pokud hráč figurku v zásobě nemá, nic se neodečte a metoda vrátí `false`.

```
/* ... */
if(x>=106 && x<=123 && y>=35 && y<=52){
  if(g.selectedCard.possibleFigures.contains(WorldSide.NEE)){
    if(g.active.getCountFigure()>0){
      for(RegionOnBoard i : g.allRegions){
        for(RegionOnCard j : i.regions){
          if(j.getCard().getX()==g.selectedCard.getX() && j.getCard().getY()==g.
              selectedCard.getY() && j.getfPosition().equals(WorldSide.NEE)){
              j.setFigure(new Figure(g.active.getColor(), WorldSide.NEE, "",
                  index, "normal", true));
              g.active.setCountFigure(g.active.getCountFigure()-1);
              return true;
}}}}}}
/* ... */
```

Výpis 1: Část původní metody AddFigure.execute

Proto jsem vytvořil abstraktní třídu, která obsahuje dvě statické metody. Jedna metoda zjistí ze souřadnic světovou stranu na kterou hráč chce umístit figurku. Druhá metoda přijímá data hry, světovou stranu a typ figurky, a figurku umístí (pokud je to možné). Tato metoda vrací hodnotu true nebo false v závislosti na tom, jestli byla figurka umístěna.

Protože i tyto třídy si byly velmi podobné, metody jsem přesunul do třídy `addFigure` a do spouštěcí metody této třídy jsem přidal parametr, který určí typ figurky. Tím se mi povedlo opět zkrátit kód a zmenšit počet tříd.

Touto úpravou se mi povedlo snížit množství kódu a počet tříd ze 4 tříd a více než tisíce řádků na zhruba sto dvacet řádků a jednu třídu.

4.3 Pole polí

Jak jsem se zmínil v předchozí kapitole, tak Java je objektově orientovaný jazyk. I přes to že pole (List a ArrayList) jsou v Javě objekty, není moc elegantní je použít v komunikaci vnořená (pole polí). V původní verzi se s tím autor ale nepáral a udělal to přesně tímto způsobem. Pro odeslání dat na server vytvořil pole, jehož obsahem bylo pole všech karet, pole změněných karet, pole hráčů a dalších několik polí. To způsobilo, že po přijetí této struktury musel program přistupovat k datům pomocí indexů. Na mě, jakožto na programátora, který má v plánu tato data rozšířit, to působilo velmi zmateně.

```
/* novy objekt StavHry */
g.packOfCards = stavHry.getPackOfCards();
g.usedCards = stavHry.getUsedCards();
g.players = stavHry.getPlayers();
g.allRegions = stavHry.getAllRegions();

/* misto nic nerikajiciho pole */
g.packOfCards = stavHry.get(0);
g.usedCards = stavHry.get(1);
g.players = stavHry.get(2);
g.allRegions = stavHry.get(3);
```

Výpis 2: Ukázka rozdílu po použití objektu StavHry

Proto jsem vytvořil objekt Stav hry, který uvnitř obsahuje každé pole, které bylo v původním velkém poli. Navíc další data, jako jsou pozice draka, víly a podobná, která by v poli být ani neměla protože tato informace je unikátní. Navíc je tento objekt přehlednější pro dalšího případného člověka, který by chtěl na aplikaci později pracovat.

4.4 Souřadnice karty

Jako velké množství programů se i tento skládá z několika vrstev. GUI je nejvyšší vrstva a neměla by obsahovat herní logiku. Stejně jako by vrstva s herní logikou neměla obsahovat data, která jsou potřebná pouze pro vykreslení (výjimkou jsou například cesty k obrázkům).

Souřadnice karet byly ale uloženy tak, aby karta mohla být vykreslena přímo na tyto souřadnice. To znamená že, když karty měly velikost 64 pixelů, souřadnice musely být dělitelné

velikostí karty. Další důležitý nedostatek byl, že karta umístěná uprostřed hrací plochy měla souřadnice [1500,1500], což postrádá v datové struktuře význam.

Rozhodl jsem se tedy souřadnicový systém karet změnit. Karta umístěná uprostřed má nyní souřadnice [0,0]. Také jsem změnil jednotky souřadnicového kříže z pixelů na kartičky. Z čehož vyplývá, že například karta vpravo od středu bude mít souřadnice [1,0] a ne [1564,1500].

Díky tomuto zjednodušení souřadnicového systému už nebylo nijak složité naprogramovat možnost zvětšit/zmenšit velikost karty.

4.5 Změna velikosti

Protože herní pole se s postupující hrou stále rozrůstá, je dobré, aby měl uživatel přehled o co největší části herní plochy. Do pravého horního rohu GUI jsem umístil posuvník, kterým uživatel může nastavit velikost karty.

Když uživatel změní posuvníkem velikost, poté co pustí posuvník, všechny kartičky se smažou a opět vykreslí v jiném měřítku. Pokud by souřadnicový systém zůstal nezměněn, pro každou kartu by se musely spočítat nové souřadnice.

4.6 Zobrazení figurek

Původně byly figurky pozicovány na kartách tak, že každá pozice figurky měla svůj obrázek. To znamenalo pro každý typ figurky sedmnáct obrázků od každé barvy a dalších sedmnáct pro zobrazení pozice možného umístění. Dohromady to tedy znamenalo 323 obrázků figurek.

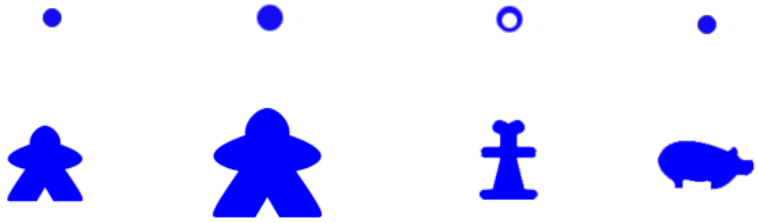
Lepší řešení by bylo, mít pro každý typ figurky od každé barvy jen jeden obrázek a na kartě mu upravit pozici. Figurky už měly pozici uloženou jako enum typu `WorldSide`. Do tohoto enumu jsem tedy přidal dvě metody `getX` (Ukázka kódu 3) a `getY`. Metody přijímají v parametrech velikost karty a figurky a vrací pozici figurky na kartě. Tyto souřadnice se počítají z vlastností `fractionForX` a `fractionForY`, které jsou zadány v parametrech konstruktoru enumu ve formě desetinného čísla.

```
/**
 * Horizontalni odsazeni figurky pro vykresleni (zleva, v pixelech)
 * @param CardSize Velikost karticky
 * @param FigureSize Velikost figurky
 * @return Odsazeni zleva
 */
public int getX(int CardSize, int FigureSize){
    return new Double((CardSize - FigureSize)*this.fractionForX).intValue();
}
```

Výpis 3: Metoda getX enumu WorldSide

Při vykreslování karty se tedy pro každou figurku zjistí, jaké má mít souřadnice na kartě a na ně se poté obrázek vykreslí. Touto optimalizací se mi povedlo snížit počet obrázků pro figurky na pouhých 28 (včetně nově přidaných figurek).

Navíc jsem místo nevzhledných barevných kruhů a koleček použil obrázky ve tvaru skutečných figurek ze hry. tyto obrázky mají také o něco vyšší rozlišení, takže po zvětšení herní plochy budou vypadat lépe.



Obrázek 8: Původní figurky (nahoře) a nové figurky (dole)

4.7 Práce s obrázky

Metody pro práci s obrázky byly rozesety různě po kódu. Vytvořil jsem proto statickou třídu ImageManager, která tyto metody seskupuje. Obsahuje pouze statické metody. Například metodu pro otočení obrázku.

Obsahuje metody i pro získání obrázku z cesty k souboru. Funguje jako jednoduchý ResourceManager a tím šetří čas aplikaci, aby nečekala na čtení z harddisku.

Dále obsahuje metodu pro získání obrázku figurky. Ať už pro figurku hráče, či figurky jako jsou drak a víla. Pokud se ale jedná o figurku hráče, musí být volána metoda s argumentem barvy. Pokud bude volána bez něj, metoda vyhodí výjimku `IllegalArgumentException`. Vnitřně tyto metody opět volají tu, která získá obrázek z cesty k souboru.

V této třídě jsem si taky vytvořil metodu, která mi sloučí obrázky pro zvýraznění podle `RegionType` a pole `WorldSide`. Abych zabránil zničení načteného obrázku, vytvořil jsem metodu `deepCopy`, která obrázek zkopíruje do nové proměnné, kterou poté vrátím. To se děje pouze u této metody.

ResourceManager uvnitř této třídy je tvořen HashMapou, která mapuje cestu k souboru na `BufferedImage`. Při volání medoty `getImage` se nejdřívě ověří, jestli soubor už nebyl načten. Pokud nebyl, načte se z harddisku a uloží se do hashmapy. Pokud už načtený byl, vrátí se pouze obrázek z paměti.

Tímto mimo jiné docílím také toho, že se nenačítají obrázky, které pro běh programu nepotřebuji.

4.8 Další možná vylepšení

Na celém projektu existuje mnoho dalších částí, které by potřebovaly upravit, vylepšit, nebo zefektivnit.

Pro přidávání rozšíření by bylo vhodné použít třídy obsahující pravidla. Tyto třídy by měly společného abstraktního předka, který by obsahoval definované metody a obsahoval popis toho, kdy se volají a jaké operace mohou provádět (podobně jako jsem vyřešil posílání vnořeného tahu). Podobné řešení by se týkalo i herních karet. Informace o rozšíření nemusí mít vždy všechny karty. Omezilo by se tak množství dat v paměti.

Určitě by bylo vhodné omezit i duplicitní úseky kódu vhodným rozdělením do znovupoužitelných tříd a funkcí (jako v případě tříd AddBuilder, AddBigFigure atd.).

5 Implementace

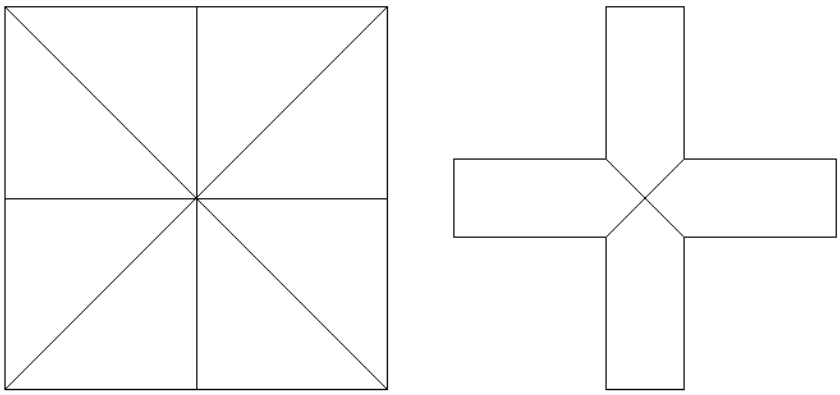
Před samotnou implementací nové herní logiky jsem musel uložit data, která ke svému fungování potřebuje. Nejdříve jsem prošel symboly na kartách, nové typy oblastí a nové figurky. Všechny tyto novinky jsem přidal do příslušných enumů.

K figurce věže bylo potřeba vymyslet složitější vykreslení, než je pouhý obrázek, protože jejich výška není nijak omezena. Zvolil jsem jednoduchý obrázek věže, který překrývá číslo, označující její výšku.

Přidávání nových karet a jejich oblastí byla velmi zdlouhavá, otrocká práce. Ještě na konci programování jsem objevoval chyby, které byly způsobeny jen nepřesnostmi v popisu kartiček. A to nejen těch, které jsem přidával nově. V odhalování těchto problémů mi hodně pomohlo zvýraznění oblastí mapy.

5.1 Zvýraznění souvislé oblasti mapy

Aby hráč viděl, kam až sahá plocha města nebo louky a jak dlouhá je cesta, implementoval jsem do GUI jednoduché zvýraznění oblastí.

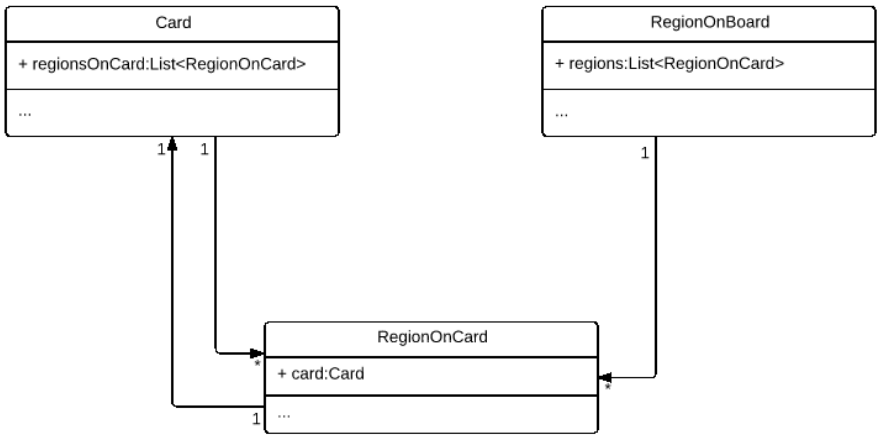


Obrázek 9: Tvar zvýraznění oblastí

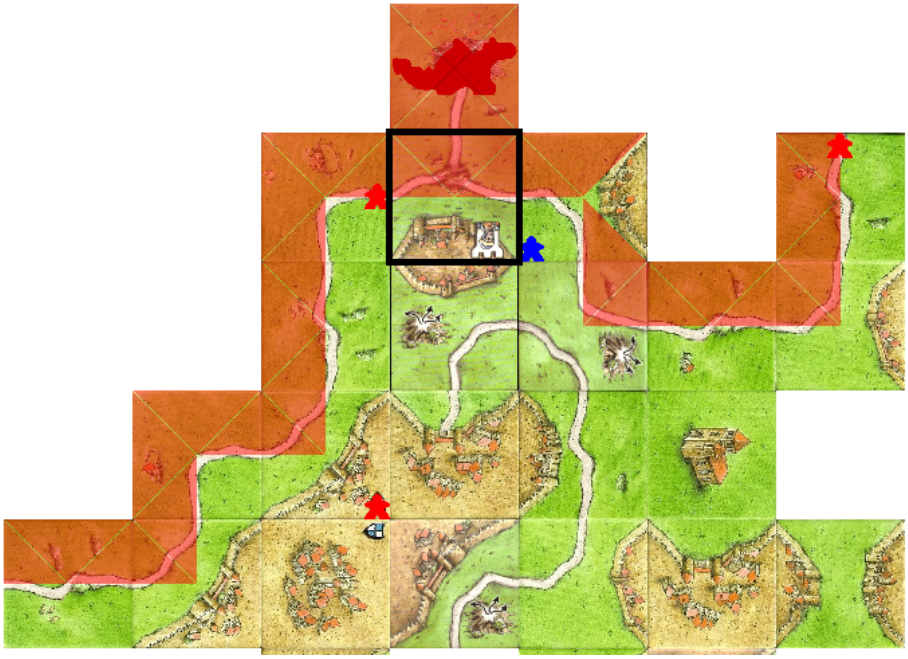
Kartičku jsem rozdělil (Obrázek 9) na osm trojúhelníků pro oblasti a čtyři cesty, podle nejčastějšího rozdělení karty. Po kliknutí na kartu v hracím plánu zjistím souřadnice kliknutí na kartě a podle nich získám oblast karty, na kterou hráč kliknul. Oblast na kartě, umístěné v herní ploše je už zařazena do většího celku (Viz diagram), který seskupuje celou jednu oblast napříč kartami. Díky tomu jsem schopný pomocí oblasti na kartě získat oblast v herní ploše, kterou si uložím.

Při překreslení herní plochy zjišťuji, zda je uložen nějaký region pro zvýraznění. Pokud ano, pro každý podregion zavolám metodu, která jej překreslí zvýrazněně.

Při použití tohoto postupu se ale vyskytl problém. Pokud na kartě bylo více oblastí součástí regionu (černě vyznačená na obrázku 11), zvýraznila se pouze jedna z nich.

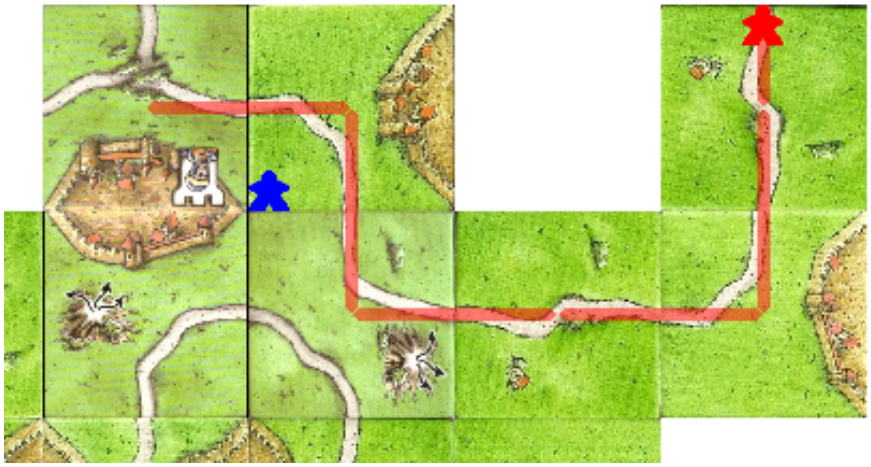


Obrázek 10: Třídní diagram uložení karet hracího pole

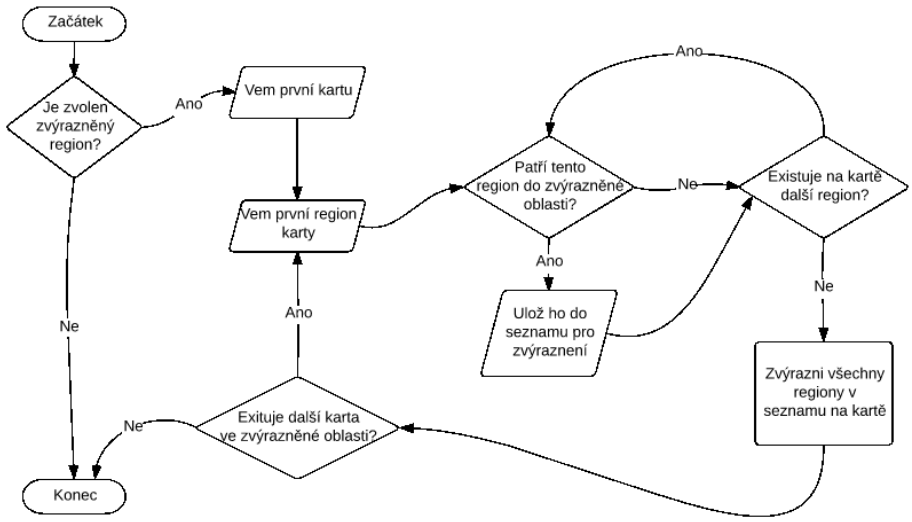


Obrázek 11: Ukázka zvýrazněné louky

Vyřešil jsem ho změnou algoritmu. Vždy pro každou část regionu zjistím kartu, na které se nachází. Poté pro všechny oblasti na kartě ověřím, jestli nejsou ve zvýrazněné oblasti. Pokud jsou, uložím si je do pole. Metodu pro vykreslení se zvýrazněnou oblastí jsem upravil, aby místo jedné oblasti přijímala pole. Vývojový diagram tohoto algoritmu můžeme vidět na obrázku 13.



Obrázek 12: Ukázka zvýrazněné cesty



Obrázek 13: Diagram algoritmu zvýraznění oblasti

5.2 Rozšíření Věže

Do hry v tomto rozšíření přibývají karty s vyobrazením základů věže, na které můžou hráči věže stavět. Počet dílků věže, který má každý hráč k dispozici, je určen počtem hráčů ve hře. Daný počet dílků se jim tedy přiřadí po založení hry. Stavěním věží do výšky mohou zajímat figurky ostatních hráčů.

5.2.1 Oblast Tower

Věž na kartě je nový typ oblasti, na který ale nemůže být umístěna figurka hráče, dokud věž není postavena (tzn. dokud nějaký hráč neumístí na kartu dílek věže). Proto jsem musel upravit algoritmus, určující na kterou část karty lze umístit figurku. Protože hrozí, že v budoucnu

budou vznikat další takovéto oblasti, rozhodl jsem se jako vlastnost enumu `RegionType` přidat informaci, jestli sem může být umístěna figurka klasickým způsobem, což je okamžitě po umístění karty do hry.

Pokud hráč chce umístit věž na kartu, klikne na příslušné tlačítko v GUI a poté zvolí kartu. Pokud na kartě je základ věže, nebo věž, která není zablokovaná figurkou, dílek se přidá a věž se zvětší.

Obrázek 14: Věž ve hře

Na věž lze umístit nejen obyčejnou figurku, ale i ostatní figurky hráče (stavitel, prasátko atd) jak můžeme vidět na obrázku 14 vpravo.

5.2.2 Zajmutí figurky

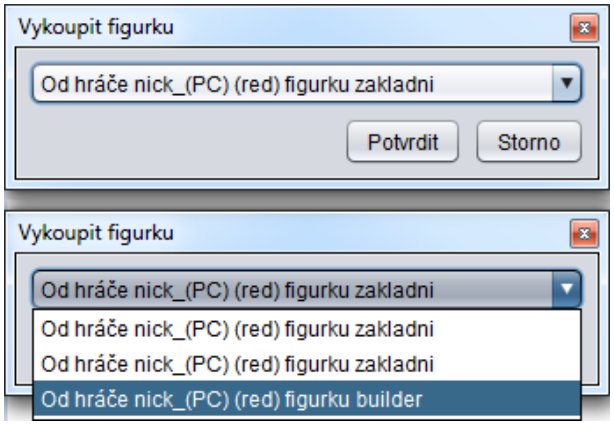
Po úspěšném umístění věže hráč může zajmout cizí figurku. Může samozřejmě zajímat i figurky na věžích. Po umístění tedy klikne na kartu (pokud je na kartě více figurek, musí kliknout přímo na figurku) a zajme figurku. Ověří se, jestli figurka nepatří jemu. Poté se uloží do pole jeho zajatých a odebere se z karty. Zajmout lze všechny figurky, které patří jinému hráči (tedy kromě víly, draka a dílků věží).

Ještě před uložením zajaté figurky do pole se ověří, zda majitel právě zajímané figurky nevlastní figurku druhého hráče. Pokud ano, figurky se obě vrátí do zásoby majitelům.

5.2.3 Vykoupení figurky

K vykupování figurek jsem vytvořil jednoduchý formulář (Obrázek 15). V tomto formuláři si hráč zvolí kterou figurku chce vykoupit a klikne na Potvrdit.

Do formuláře se po jeho vytvoření pošlou data a ve formuláři se pak zpracují. Projde všechny hráče a jejich zajaté figurky a do výběru zařadí všechny nalezené figurky patřící hráči, který chce figurku vykoupit.



Obrázek 15: Formulář pro vykoupení figurky

5.3 Rozšíření Princezna a drak

Do hry jsou přidány figurky draka a víly. Drak figurky vrací z hrací plochy zpět majitelům a víla naopak figurky před drakem ochraňuje. Víla navíc umožňuje získat i herní body. Další novinkou jsou symboly princezny a tajné chodby. Princezna může vyhostit rytíře z města, tajnou chodbou se dokáže figurka dostat na různá území po celém herním plánu.

5.3.1 Princezna ve městě

Poté co hráč vylosuje kartu se symbolem princezny ve věži, umístí ji běžným způsobem na hrací plochu. Pokud ji položí ke kartě s městem, ve kterém stojí jedna nebo více figurek, musí jednu z nich (podle svého výběru) vrátit do zásoby majiteli. On sám nesmí v tomto případě umístit žádnou svou figurku. [2]

Při tom, když je karta umístěna na hrací plochu, program ověří, jestli na kartě je region se symbolem princezny. Pokud ano, zjistí, jestli se po přidání do hry tento region připojil k již existujícímu městu, ve kterém je nějaká figurka. Pokud se zjistí že ano, uloží se tento region z kartičky do speciální proměnné.

Pokud po umístění kartičky program zjistí, že byl uložen region s princeznou, zjistí, jaké oblasti na herní ploše je součástí, zakáže všechny možnosti umístění figurky a vynuluje možnosti, pro umístění figurky na přidanou kartu. Poté čeká, než hráč klikne na kartu města, na které je figurka k vyhoštění.

Po kliknutí a vyhoštění figurky je okamžitě odeslán na server konec tahu hráče, protože v tomto tahu už nemůže provést žádné akce.

5.3.2 Tajné chodby

Na některých kartách tohoto rozšíření je symbol tajné chodby. Po umístění této karty může hráč zvolit její použití. V tu chvíli může hráč umístit figurku na jakékoliv neuzavřené a neobsazené území na herní ploše kromě louky.

Pokud je tedy tento symbol na kartě, po jejím umístění se uloží, že lze tuto možnost použít. Po kliknutí na tlačítko "Tajná chodba"a následném kliknutí na kartu v herním plánu se do proměnné s právě vybranou kartou uloží karta, na kterou bylo kliknuto. Zároveň se pro ni znovu spočítá, na které části karty mohou být umístěny figurky.

Metodu, která možné umístění počítá, jsem tedy upravil, aby přijímala další parametr "viaTunel". Je to `boolean` proměnná, která označuje, jestli je metoda zavolána při použití tajné chodby. V případě že ano, se ověří, jestli na této kartě není drak a nepřidá možnosti umístění figurky na louku.

Abych nemusel měnit předchozí kód, udělal jsem druhou metodu s původními argumenty. Ta jen zavolá metodu se všemi argumenty, přičemž do chybějícího argumentu pošle hodnotu false (Ukázka kódu 4).

```
public class PossibleFigures {
    public static void execute(Card card,PravidlaHry g){
        PossibleFigures.execute(card, g, false);
    }
    public static void execute(Card card,PravidlaHry g, boolean viaTunel){
      /* ... */
    }
}
```

Výpis 4: Přemosťující metoda třídy PossibleFigures

5.3.3 Umístění figurek víly a draka

Protože víla i drak jsou unikátní figurky, místo abych přiřadil tuto figurku kartě, uložím si kartu, na které se figurka nachází. To mi ulehčí její přesouvání a vykreslení. Kdyby tyto figurky byly uloženy v kartě, musel bych při každém přesunu procházet všechny karty na ploše a zjišťovat, na které z nich figurka stojí. Nyní při přesunu figurky nejdříve uložím původní pozici do pomocné proměnné, do proměnné pozice figurky přiřadím kartu, na kterou se figurka přesunula a poté obě karty překreslím.

5.3.4 Pohyb draka

Pohyb draka je sám o sobě typický. Pokud dojde k tomuto tahu, každý hráč pohne drakem (první hráč, který pohyb způsobil a dále ve směru hry). Poté co drak skončí svůj pohyb, pokračuje ve hře hráč, který následuje po iniciátorovi pohybu draka. Jedná se tedy o vložený tah, který neovlivní další pořadí hráčů.

Vzhledem k tomu, že více rozšíření obsahuje takovéto vnořené tahy, jsem se rozhodl vytvořit abstraktní třídu `ExtendedMove`. Tato třída bude moct být použita pro další různé vnořené tahy s daleko menším zásahem do ostatního kódu. Inspiroval jsem se v knize o návrhových vzorech [5].

```
public abstract void afterMoveReceiving(PravidlaHry g);
public abstract void beforeNext(PravidlaHry g);
public abstract List<Card> beforeEnd(PravidlaHry g);
public abstract void cardClicked(Card c);
public abstract String getNextNick();
```

Výpis 5: Abstraktní metody třídy ExtendedMove

Všechny tyto metody musí obsahovat jakýkoliv potomek této třídy. Metody se volají v konkrétních místech tahu. Ne všechny z těchto metod musí obsahovat kód, ovšem metody musí existovat, protože se volají automatizovaně takto:

- afterMoveReceiving - na začátku, okamžitě poté, co hráč obdrží tah
- beforeNext - před tím, než se odešle tah dalšímu hráči
- beforeEnd - před ukončením vnořeného tahu, navíc vrací pole karet, které je nutno překreslit
- cardClicked - při kliknutí na kartu
- getNextNick - tato metoda by měla vrátit jméno hráče, který je další na řadě

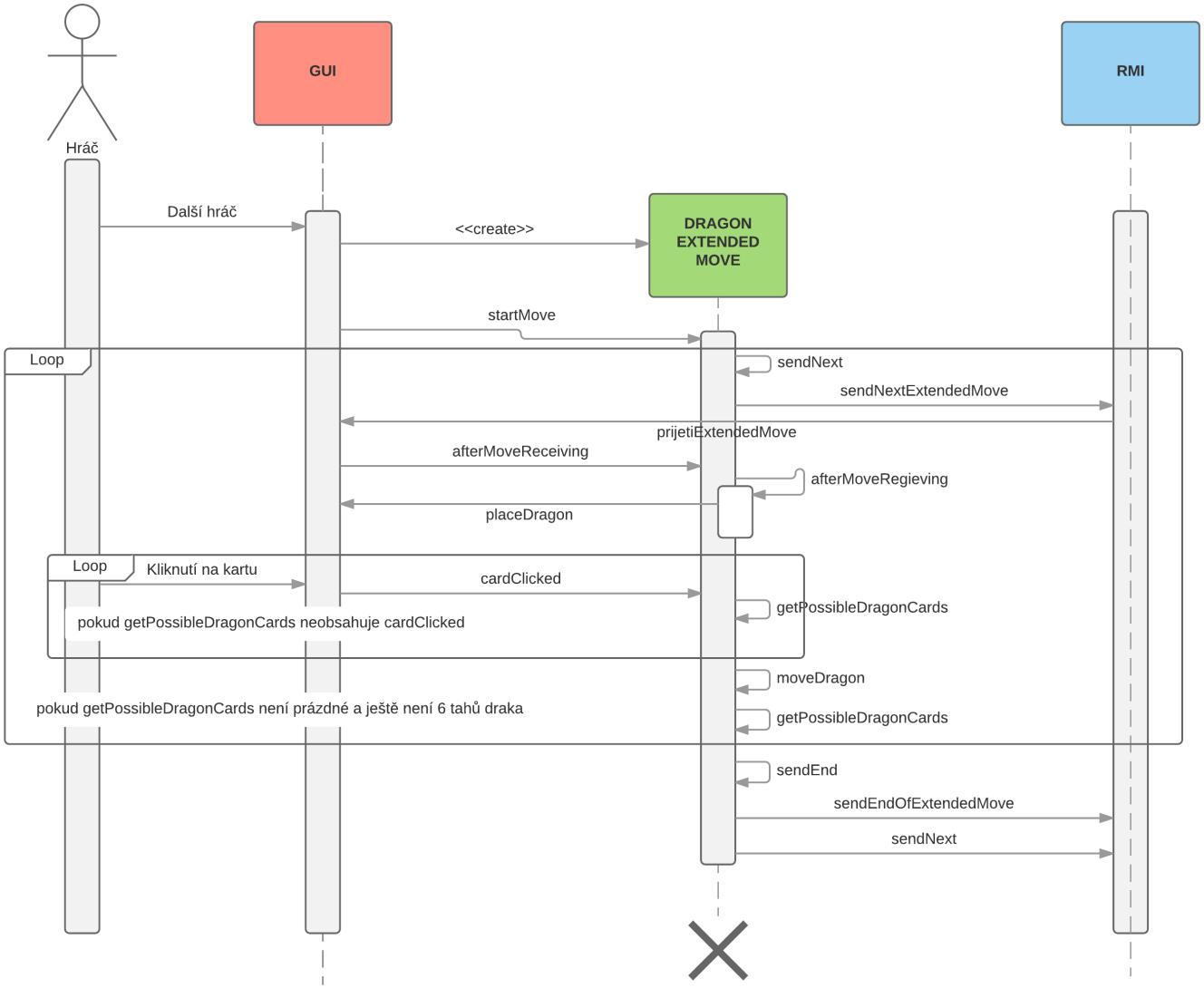
Samozřejmě tyto metody nemusí dostačovat pro použití u jiného vnořeného tahu, ale minimálně ulehčí práci s tím spojenou.

Dále abstraktní třída obsahuje několik předdefinovaných metod. Tyto metody mohou využívat všichni potomci.

```
public void startMove() throws RemoteException{/* kod */}
protected void sendNext() throws RemoteException{/* kod */}
protected void sendEnd() throws RemoteException{/* kod */}
```

Výpis 6: Implementované metody třídy ExtendedMove

- startMove - po vytvoření vnořeného tahu musí program zavolat tuto metodu, aby se vnořený tah odeslal
- sendNext - pokud hráč skončil, pošle tah dalšímu hráči
- sendEnd - tah skončil, hra pokračuje normálním způsobem



Obrázek 16: Sekvenční diagram pohybu draka

6 Analýza systémem SonarQube

Analýza tímto softwarem ukázala, že ačkoliv stávající verze projektu obsahuje více funkčnosti, klesla jeho složitost. Fyzicky v celém projektu přibylo necelých tisíc řádků kódu, ovšem bylo odstraněno 1600 duplicitních řádků (viz obrázek 17). Z obrázku můžeme také vyčíst, že klesl počet problémových částí kódu. Samozřejmě kód, který jsem v proběhu programování smazal jsem nahradil jiným, se stejnou funkčností.

	Apr 27 2016 1.0	Apr 27 2016 2.0
Issues	1,536	1,269
Blocker issues	1	1
Critical issues	50	38
Major issues	1,077	826
Minor issues	391	387
Info issues	17	17
Technical Debt	38d	31d

	Apr 27 2016 1.0	Apr 27 2016 2.0
Duplicated lines (%)	19.9%	7.1%
Duplicated lines	2,615	1,039
Duplicated blocks	291	255
Duplicated files	28	18

	Apr 27 2016 1.0	Apr 27 2016 2.0
Complexity	2,779	2,556
Complexity /function	5.7	4.3
Complexity /class	32.7	27.2
Complexity /file	33.9	28.1

Obrázek 17: Část výstupu z analýzy systémem SonarQube (verze 1.0 je původní, verze 2.0 je nynější)

7 Závěr

Cílem této práce bylo rozšířit stávající verzi hry Carcassonne v jazyce Java o další dvě rozšíření (Věže, Princezna a drak) a při zapnutí hry libovolně kombinovat použitá rozšíření. Dále bylo úkolem umožňovat zvýraznění souvislé oblasti mapy (cesta, město, louka) a pro přihlášení využít nově vznikající herní portál.

Bohužel herní portál není k dispozici, takže tuto část jsem nemohl splnit.

Ostatní části zadání jsou ale splněny. Jsou implementována hratelná rozšíření, zvýrazňování oblastí a upraveno vykreslování karet. Také velká část předchozího kódu je upravena do lépe pochopitelné podoby.

Literatura

[1] Pravidla hry Carcassonne In: *SVĚT-HER.CZ* [online] [cit. 2016-04-15]
Dostupné z: `http://www.svet-her.cz/soubory/377/Carcassonne_cze.pdf`

[2] Pravidla rozšíření Princezna a drak In: *Hrajeme.cz* [online] [cit. 2016-04-15]
Dostupné z: `http://old.hrajeme.cz/Hrajeme/Files/princezna.pdf`

[3] Pravidla rozšíření Věže In: *Hrajeme.cz* [online] [cit. 2016-04-15]
Dostupné z: `http://old.hrajeme.cz/Hrajeme/Files/vez.pdf`

[4] Carcassonne (hra) In: *Wikipedia: the free encyclopedia* [online] San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2016-04-15]
Dostupné z: `https://cs.wikipedia.org/wiki/Carcassonne_(hra)`

[5] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides (Gang of Four): Návrh programů pomocí vzorů. Grada. Praha 2003. ISBN 8024703025

A CD

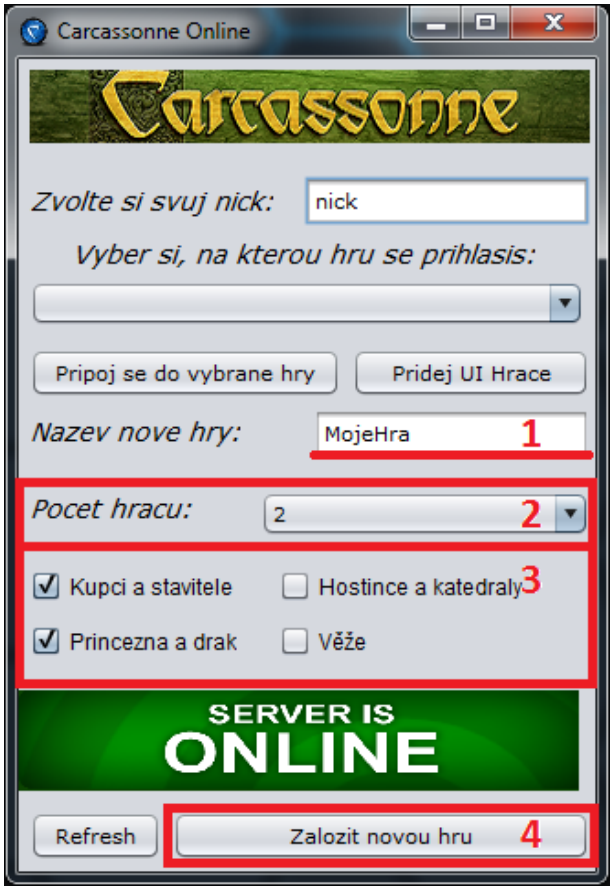
Disk obsahuje:

- Elektronickou verzi tohoto dokumentu
- Zdrojové kódy hry a serveru

B Uživatelská dokumentace

B.1 Založení hry

Založení hry je opravdu jednoduché. Postupujte podle obrázku 18.



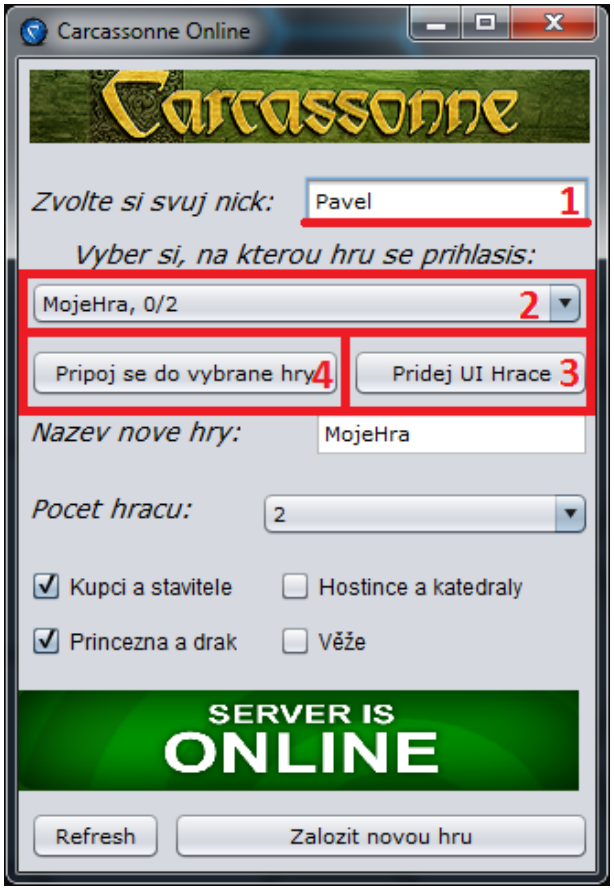
Obrázek 18: Založení hry

1. Do tohoto pole napište název vaší hry
2. Zvolte si počet hráčů
3. Zvolte, která rozšíření budou ve hře použita
4. Kliknutím založíte hru

B.2 Připojení do hry

Když je hra založena, můžeme se připojit (Obrázek 19).

1. Vyplňte svou přezdívku ve hře



Obrázek 19: Připojení do hry

2. Zvolte si, do které hry se chcete připojit
3. Pokud nemáte s kým hrát, můžete přidat automatického hráče
4. Kliknutím se připojíte do hry

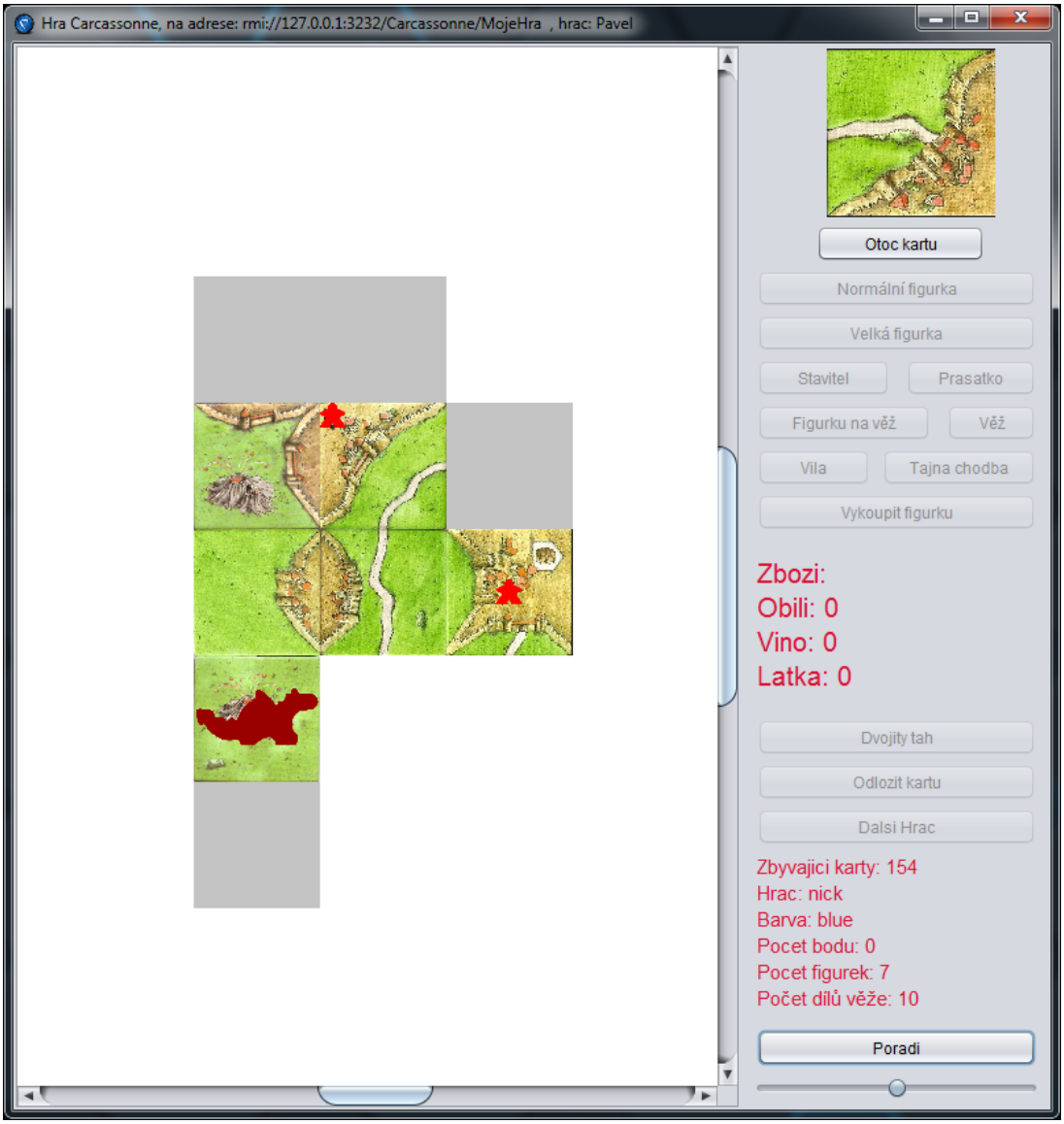
B.3 Herní rozhraní

Po připojení do hry se objeví herní rozhraní (Obrázek 20). Některé prvky nemusí být zobrazeny, v závislosti na tom, jaká jsou zvolena rozšíření.

Bílá plocha s kartičkami vlevo je herní plocha, šedé čtverce označují možné umístění karty. Kartu, kterou právě umisťujeme můžeme vidět v pravém horním rohu. Kartu lze otočit tlačítkem pod náhledem.

V pravém dolním rohu vidíme posuvník, který mění velikost karet. Jeho posunutím doprava karty zvětšíme, doleva naopak zmenšíme.

Tlačítko nad posuvníkem zobrazí aktuální pořadí hráčů.



Obrázek 20: Herní rozhraní

B.3.1 Akce hráče

Na obrázku 21 můžeme vidět, jak vypadá náhled karty a tlačítka po umístění figurek. Tlačítka nemusí být povolena, pokud hráč danou akci nemůže provést (například nemá figurky, nebo to aktuální stav hry jinak nedovoluje).

Na náhledu karty můžeme také vidět, kam je možné umístit figurku. Umístíme ji jednoduše kliknutím na bílou figurku v náhledu.

Pokud chceme umístit jinou figurku než obyčejnou, klikneme na jedno z tlačítek "Velká figurka", "Stavitel", "Prasatko".

Pro umístění věže, nebo figurky na věž klikneme na příslušné tlačítko a poté na kartičku s věží na herním plánu. Pokud umístíme dílek věže, je okamžitě možné zajmout figurku některého

Obrázek 21: Umístění figurky

protihráče. Prostě na ni klikneme v hracím poli. Nebo můžeme kliknutím na tlačítko "Další hráč"ukončit tah.

Pro umístění víly opět klikneme na příslušné tlačítko a poté na herním plánu zvolíme kartu pro její nové umístění.

Pokud kartička obsahuje tajnou chodbu, bude povoleno tlačítko "Tajná chodba". Po kliknutí na něj zvolíme v herním plánu kartičku, na kterou chceme figurku umístit. Ta se zobrazí v náhledu a postupujeme jako bychom chtěli klasicky umístit figurku.

Tlačítko "Vykoupit figurku"slouží při hře s rozšířením Věže k vykoupení zajaté figurky. Po kliknutí na něj se zobrazí okénko, ve kterém zvolíme kterou figurku a od kterého hráče můžeme vykoupit.

B.3.2 Akce vázané na umístění karet

Když umístíme kartu s princeznou ve městě do hradu, ve kterém je rytíř, není možné provést jakoukoliv jinou akci, kromě vyhoštění rytíře. Rytíře vyhodíme z hradu tak, že na něj klikneme v hracím plánu.

Když nastane pohyb draka, hráči s ním pohybují tak, že klikají na kartu, na kterou chtějí aby drak popošel.

B.3.3 Další akce hráče

Pokud umístíme kartu tak, že rozšíří území, na kterém je figurka stavitele, můžeme provést dvojitý tah. To provedeme kliknutím na tlačítko "Dvojitý tah".

Pokud není možné kartu do hry umístit, stiskneme tlačítko "Odložit kartu". Tím se karta zahodí a hráč dostane novou.